Delivering Content via Interactive QuickTime

Session 604

Delivering Content via Interactive QuickTime

Tim Monroe
QuickTime Engineering

What You Will Learn

- How to create interactive QuickTime movies
- How to use QuickTime APIs to enhance interactivity in your QuickTime application
- How to make your QuickTime application “interactive-savvy”

What Is Interactivity?

- Objects in a movie respond to mouse and button events
- Different media types can control one another
- Objects are “scriptable”
- Objects can communicate with other objects (even with objects in other movies)

Interactivity Roadmap

- Some track types are already interactive: QuickTime VR, Flash
- Some track types can be made interactive by being “wired”: Sprite, Text, QuickTime VR, Flash
- We can use QuickTime APIs and wiring to manipulate properties of noninteractive tracks

Interactivity Support

- “Interactive Challenged”
- “Interactive Aware”
- “Interactive Smart”
- “Interactive Genius”

Interactive Challenged

- Application does not use a movie controller
- Application tries to be too clever:

```
if (GetMovieRate(myMovie) != 0)
        MCIsPlayerEvent(myMC, &myEvent);
```

Interactive Aware

- Use a movie controller
- Install a movie controller action filter procedure to handle movie controller actions

Interactive Smart

- Support key-up events
- Support fullscreen playback
- Support intermovie communication
- Support application messages

Interactive Genius

- Support skinned movies
- Support contextual pop-up menus

Demo

Interactive Genius:
Skins and Contextual Menus

Interactive Aware

- Install a movie controller action filter procedure:

```
myFilterUPP = NewMCActionFilterWithRefConUPP
                        (MyMCActionFilterProc);
MCSetActionFilterWithRefCon(myMC, myFilterUPP,
                        (long)myWindowData);
```

Interactive Aware

- Handle movie controller actions:

```
pascal Boolean MyMCActionFilterProc
      (MovieController theMC, short theAction,
       void *theParams, long theRefCon)
{
      switch (theAction) {
            case mcActionControllerSizeChanged:
                  MyResizeWindow(theRefCon);
                  break;
      }
      return(false);
}
```

Interactive Smart

- Support key-up event *(new in QT 6.0)*
  - If using **WaitNextEvent**, call **SetEventMask(everyEvent)**
  - If using Carbon events, handle **kEventRawKeyUp**
  - Just give the event to the movie controller . . .

Interactive Smart

- Support fullscreen playback:
  BeginFullScreen
  EndFullScreen
  - Adjust movie controller settings
- Support fullscreen triggers:
  - Look for user data item of type ‘ptv ’
  - Handle application messages

Demo

Interactive Smart:
Fullscreen Playback

Programmed Interactivity

- Use QuickTime APIs to add interactive behaviors to movies
- Interactive logic resides in the playback application

Demo

Autospinning VR

Autospinning VR

- Install a timing mechanism

```
InstallEventLoopTimer(GetMainEventLoop(), 0,
    TicksToEventTime(kDelayTicks),
    gWinTimerHandlerUPP, (void *)myInstance,
    &myTimerRef);
```

Autospinning VR

- Adjust pan angle when timer fires

```
pascal void MySpinTimer
        (EventLoopTimerRef theTimer, void *theRefCon)
{
    QTVRInstance    myInstance =
                            (QTVRInstance)theRefCon;
    float           myPanAngle;

    myPanAngle = QTVRGetPanAngle(myInstance) + 1;
    if (gOkayToAutoSpin)
        QTVRSetPanAngle(myInstance, myPanAngle);
}
```

Autospinning VR

- Stop spinning if user clicks in movie

```
case mcActionMouseDown:
    gOkayToAutoSpin = false;
    InstallEventLoopTimer(…);
    break;
```

- Start spinning if no clicks in 10 seconds

```
pascal void MyRestartSpinTimer
    (EventLoopTimerRef theTimer, void *theRefCon)
{
    gOkayToAutoSpin = true;
}
```

Demo

VR Node Transitions

VR Node Transitions

- Use QuickTime VR callback functions
  - Node-leaving procedure
  - Node-entering procedure

VR Node Transitions

- In node-leaving procedure: Draw the source node picture into an offscreen graphics world

```
GetMovieBox(myMovie, &myRect);
NewGWorld(&myTargetGWorld, 0, &myRect, 0, 0, 0L);
SetMovieGWorld(myMovie, myTargetGWorld, NULL);
MoviesTask(myMovie, 0L);
```

VR Node Transitions

- In node-entering procedure: Draw the destination node into an offscreen graphics world and run the effect

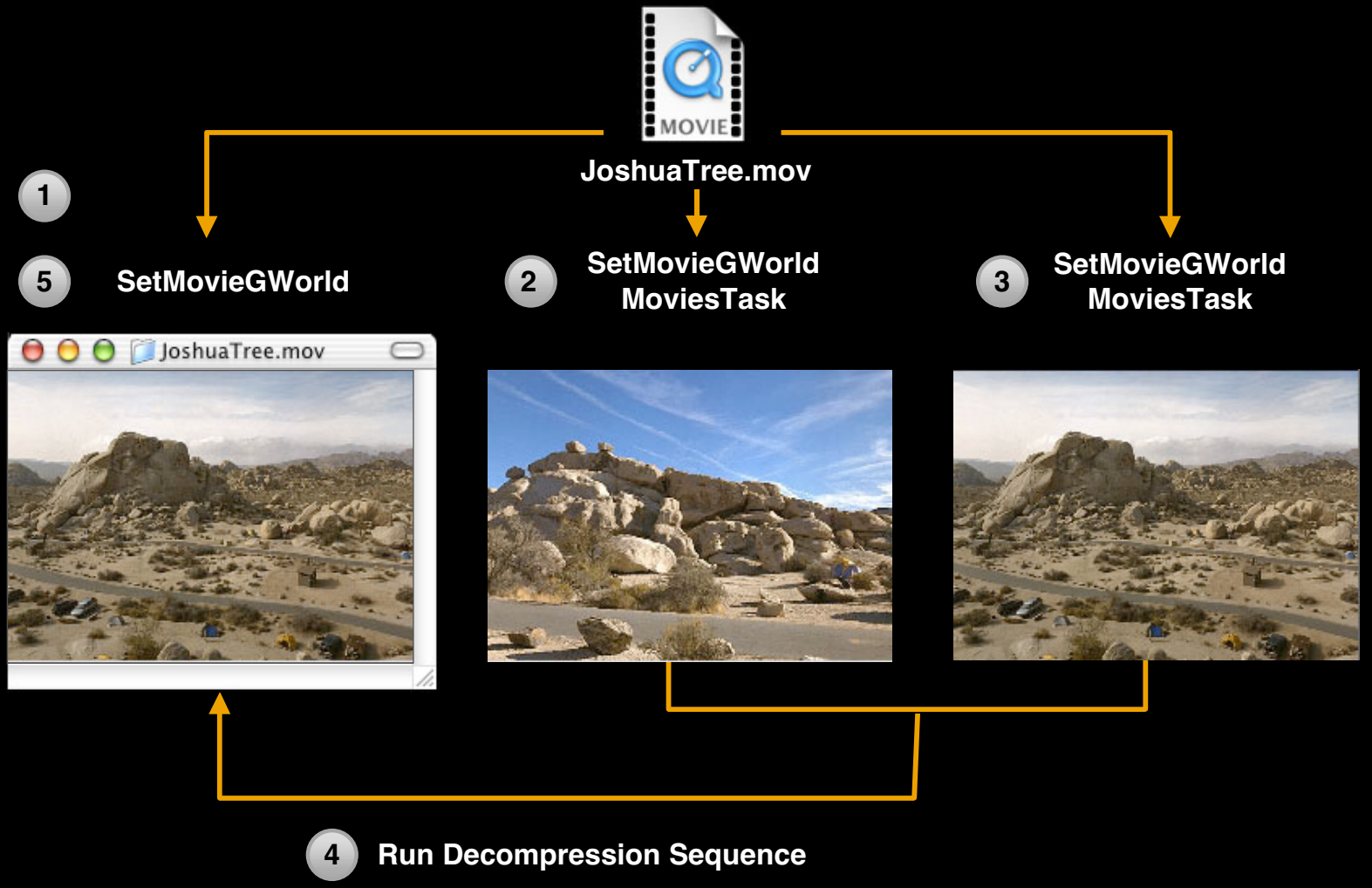
```
NewGWorld(&myTargetGWorld, 0, &myRect, 0, 0, 0L);
SetMovieGWorld(myMovie, myTargetGWorld, NULL);
MoviesTask(myMovie, 0L);

// run effect sequence
MyRunTransitionEffect(myInstance);
```

VR Node Transitions

- In node-entering procedure: When effect sequence is done, reset movie and controller graphics worlds

```
MCSetControllerPort(myMC, myPort);
SetMovieGWorld(myMovie, myPort, NULL);
MCMovieChanged(myMC, myMovie);
```

VR Node Transitions



Demo

Time-Jump Transitions

Time-Jump Transitions

- Handle time-jump actions:

```
pascal Boolean MyMCActionFilterProc
      (MovieController theMC, short theAction,
       void *theParams, long theRefCon)
{     switch (theAction) {
            case mcActionGoToTime:
                  DoTimeJump(theRefCon, theParams);
                  break;
      }
      return(false);
}
```

Other Goodies

- Use intercept procedures to add directional sounds to QTVR movies
- Use time base callback functions to trigger actions at specific times in a linear movie
- Use text-loading procedures to get the text in a text sample
- Etc.

Limitations of Using APIs

- Using QuickTime APIs to add interactive behaviors to tracks requires a *special playback application*
- For some uses, this might not be an issue:
    - Standalone kiosks
    - CD-ROM-based titles

The Solutions

- Move the interactivity from the application into the media: script the media
- Write a custom media handler
    - Retains speed of compiled code
    - Can use component download program

Scripted Interactivity

- The movie itself contains information about the interactive behaviors
- So, the movie can be deployed without a special playback application:
    - QuickTime Player
    - QuickTime web browser plug-in
    - Any QuickTime-savvy application

Scripting

- Flash provides ActionScript
- QuickTime provides “wired actions”

Demo

Flash ActionScript

ActionScript

- Attach actions to buttons and movie clips

```
on (press) {
        startDrag(“/Nose2”);
}
on (release) {
        stopDrag();
}
```

ActionScript

```
onClipEvent (enterFrame) {
        _rotation = 0;
        dx = _x - _root._xmouse;
        dy = _y - _root._ymouse;
        _width = Math.sqrt(dx * dx + dy * dy);
        r = Math.atan2(dy, dx) * 180 / 3.1415926;
        if (r <= 0) {
                r = r + 360;
        }
        _rotation = r;
}
```

Action Wiring

- A *wired atom* indicates which event triggers the atom and what actions to perform when the event occurs
- A *wired track* is a track that contains one or more wired atoms

Action Wiring

- Wired atoms can use:
  - Constants
  - Variables
  - Looping
  - Branching
  - Operands

Wired Atoms


kQTEventType
event type
kAction
1
kWhichAction
1
Action type
kActionParameter
1
Parameter data

Wired Atoms

```
QTAtomContainer MyBuildURLAtomContainer (void)
{
    QTAtomContainer  myContainer = NULL;
    QTAtom                  myEventAtom, myActionAtom;
    char                    myURL[] = “www.apple.com”;

    QTNewAtomContainer(&myContainer);
    QTInsertChild(myContainer,
        kParentAtomIsContainer, kQTEventType,
        kQTEventMouseClick, 0, 0, NULL,
        &myEventAtom);
    QTInsertChild(myContainer, myEventAtom, kAction,
        1, 1, 0, NULL, &myActionAtom);
```

Wired Atoms

```
    myAction = EndianS32_NtoB(kActionGoToURL);
    QTInsertChild(myContainer, myActionAtom,
        kWhichAction, 1, 1, sizeof(long), &myAction,
        NULL);

    QTInsertChild(myContainer, myActionAtom,
        kActionParameter, 1, 1, strlen(myURL) + 1,
        myURL, NULL);

    return(myContainer);
}
```

Wired Sprites

- A keyframe for a sprite is an atom container that contains:
    - Shared data atom (sprite images)
    - Sprite atoms
        - Image index atom
        - Layer atom
        - Matrix atom
        - . . .
        - Event atom

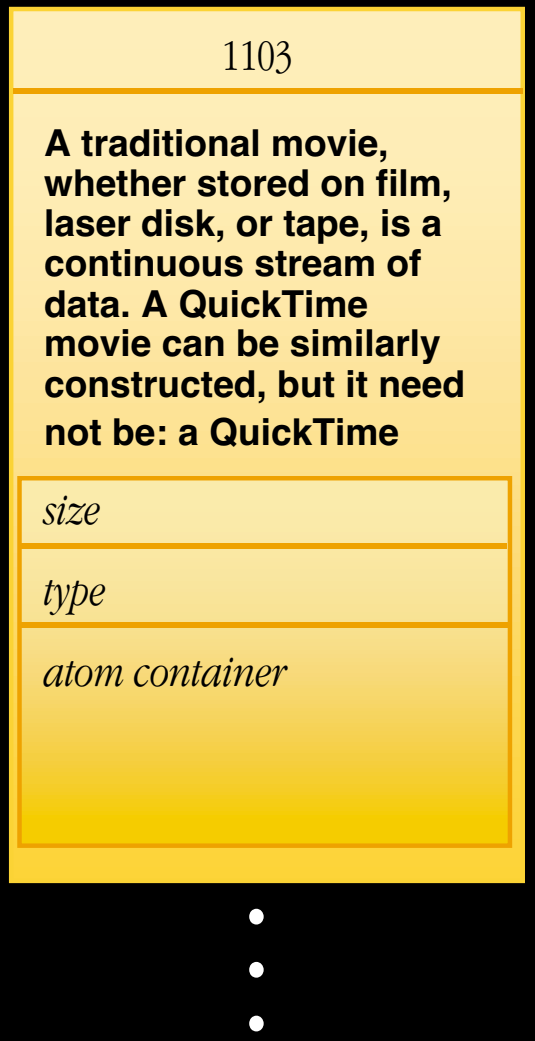
Demo

Wired Sprites

Wired Text

- A text media sample can have one or more *text atom extensions*
- There is a wired text atom extension
- Wired text atom can contain one or more *hypertext item atom* for text ranges
- Hypertext item atom contains an event atom

Wired Text Sample

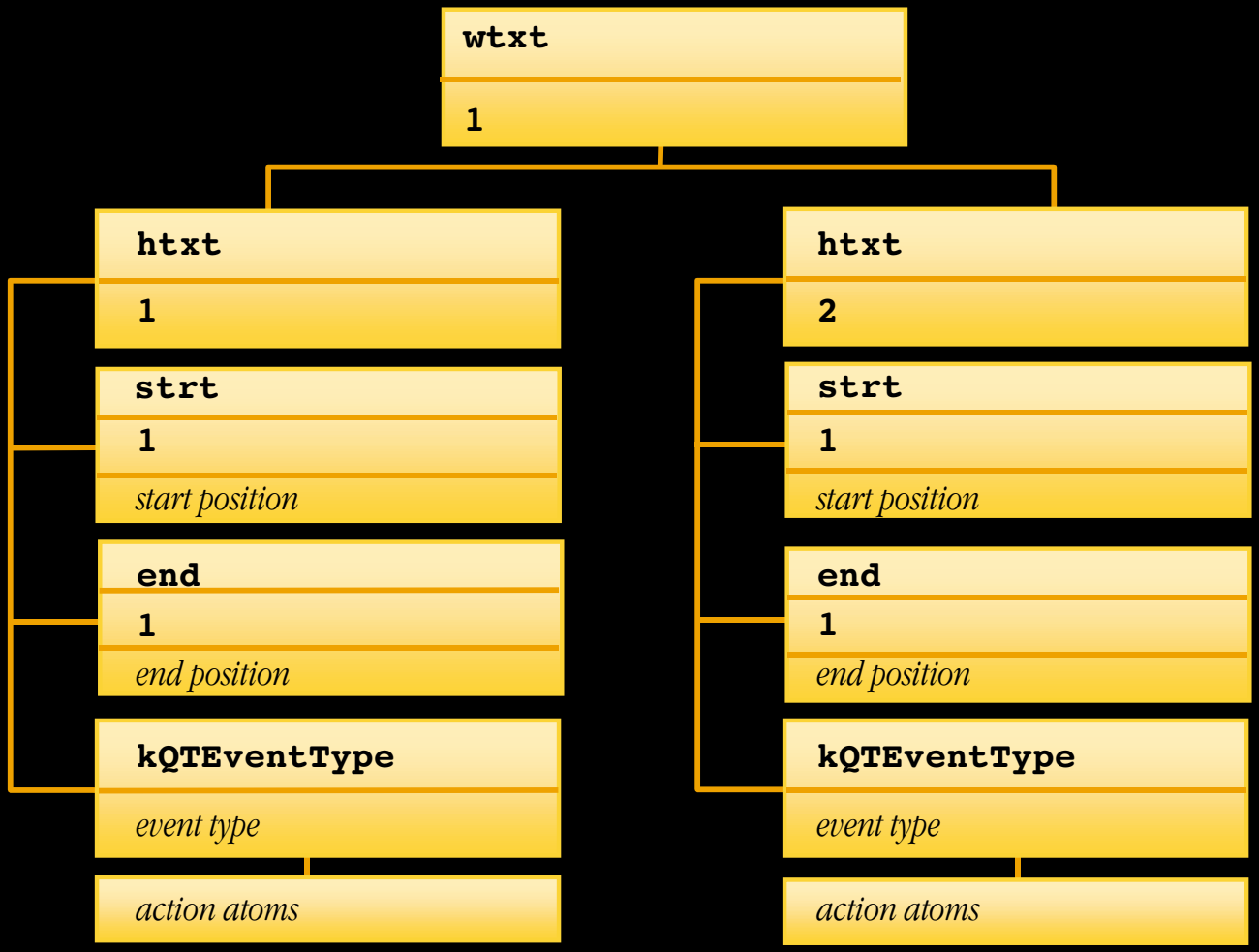


Number of bytes of text

Text

Text atom extension

Text Atom Extension



Demo

Wired Text

Wired Flash

- Flash buttons and frames contain *action lists*
- Each action in an action list has a tag
- The tag 0xAA is reserved for QuickTime wired actions
- The data for tag 0xAA is an atom container holding wired actions

Demo

Wired Flash

Wired QuickTime VR

- We can associate wired actions with a *node* or with a *hot spot*
- Node wired actions are put into a node information atom container inside the media sample
- Hot spot wired actions are put into a hot spot atom inside of a hot spot parent atom inside the media sample

Demo

Wired VR

Autospin Wired Atom

```
QTAtomContainer     myContainer = NULL;
QTAtom              myEventAtom, myActionAtom;
long                myAction;
float               myPanAngle = 1.0;
UInt32              myFlags = kActionFlagActionIsDelta
                        | kActionFlagParameterWrapsAround;

QTNewAtomContainer(&myContainer);
QTInsertChild(myContainer, kParentAtomIsContainer,
   kQTEventIdle, 1, 1, 0, NULL, &myEventAtom);
QTInsertChild(myContainer, myEventAtom, kAction,
   1, 1, 0, NULL, &myActionAtom);
```

Autospin Wired Atom

```
myAction =
    EndianU32_NtoB(kActionQTVRSetPanAngle);
QTInsertChild(myContainer, myActionAtom,
    kWhichAction, 1, 1, sizeof(long), &myAction, NULL);

myPanAngle =
    MyConvertFloatToBigEndian(&myPanAngle);
QTInsertChild(myContainer, myActionAtom,
    kActionParameter, 1, 1, sizeof(float), &myPanAngle,
    NULL);
myFlags = EndianU32_NtoB(myFlags);
QTInsertChild(myContainer, myActionAtom,
    kActionFlags, 1, 1, sizeof(UInt32), &myFlags, NULL);
```

Action Targets

- Wired actions have a default target
- You can specify some other target by adding a target atom
- You can even target *another movie*

Demo

Interactive Smart:
Intermovie Communication

Interactive Smart

- Support intermovie communication:

```
switch (theAction) {
    case mcActionGetExternalMovie:
        MyGetExternalMovie(theRefCon, theParams);
        break;
    case mcActionGetMovieName:
        MyGetMovieName(theRefCon, theParams);
        break;
    case mcActionGetMovieID:
        MyGetMovieID(theRefCon, theParams);
        break;
}
```

Application Messages

- Provide a way for interactive media to communicate with your application
    - QuickTime application messages
    - Flash application messages

QuickTime Messages

- Available in QuickTime 5.0.1 and later
- Usually generated by a wired atom:
  kActionSendAppMessage
- Intercepted in movie controller action filter proc:
  mcActionAppMessageReceived
- QuickTime defines five messages
- Apps can define their own custom messages

Application Messages

```
switch (theAction) {
        case mcActionAppMessageReceived:
                switch ((long)theParams) {
                        case kQTAppMessageEnterFullScreenRequested:
                                MyEnterFullscreen(theRefCon);
                                break;
                        case kQTAppMessageExitFullScreenRequested:
                                MyExitFullscreen(theRefCon);
                                break;
                        case kQTAppMessageWindowCloseRequested:
                                // be careful here!
                                MyCloseWindow(theRefCon);
                                break;
                }
        break;
}
```

Demo

QuickTime Application Messages

Flash Messages

- ActionScript can execute FSCommands
- Flash media handler converts them into movie controller actions of type mcActionDoScript
- Associated data includes two text strings, which your application can interpret as it desires

Interactive Smart

- Support Flash application messages

```
QTDoScriptPointer myScript =
                        (QTDoScriptPointer)theParams;

switch (theAction) {
        case mcActionDoScript:
                MyDoFSCommand(myScript, theRefCon);
                break;
}
```

Interactive Smart

- Support Flash application messages

```
void MyDoFSCommand (QTDoScriptPointer theScript,
                                        void *theRefCon)
{
        if (strcmp(myScript->command, “fullscreen”) == 0) {
                if (strcmp(myScript->arguments, “true”) == 0)
                        MyEnterFullscreen(theRefCon);
                if (strcmp(myScript->arguments, “false”) == 0)
                        MyExitFullscreen(theRefCon);
        }
}
```

Demo

Flash Application Messages

Interactive Genius

- Support skinned movies
  - A skinned movie is a movie with a custom window shape
  - An application needs special code to handle skinned windows

Interactive Genius

- Retrieving skin data:

```
myTrack = GetMovieIndTrackType(myMovie, 1,
        FOUR_CHAR_CODE('skin'), movieTrackCharacteristic);
myMedia = GetTrackMedia(myTrack);
myHandler = GetMediaHandler(myMedia);

MediaGetPublicInfo(myHandler,
        FOUR_CHAR_CODE('skcr'), myPicture, NULL);

MediaGetPublicInfo(myHandler,
        FOUR_CHAR_CODE('skdr'), myPicture, NULL);
```

Interactive Genius

- Support Flash contextual pop-up menu

```
EventRecord *myEvent = (EventRecord *)theParams;

switch (theAction) {
        case mcActionMouseDown:
                if (myEvent->modifiers & controlKey)
                        MyShowFlashContextMenu(myEvent);
                break;
}
```

Interactive-Savvy Applications

- *Use a movie controller to manage movies*
- *Support intermovie communication*
- *Support key-up event*
- *Handle application messages*
- *Support skinned movies*
- *Support fullscreen playback*

Bringing It All Together

- Export your app's interactive behaviors as wired actions
- Exploit ActionScript in Flash tracks
- Use a QuickTime movie as the front-end for your application

QuickTime Roadmap

600 The State of QuickTime in 2002	Room A2 **Wed., 9:00am**
601 Building QuickTime-Savvy Apps	Room A2 **Wed., 10:30am**
602 QuickTime for Video-Intensive Applications	Room A2 **Wed., 2:00pm**
603 Media Integration With QuickTime	Room A2 **Wed., 3:30pm**
604 Delivering Content via Interactive QuickTime	Room A2 **Wed., 5:00pm**

QuickTime Roadmap (Cont.)

FF010 QuickTime	Room J1 **Fri., 10:30am**
606 QuickTime for the Web	Room A2 **Fri., 2:00pm**
607 QuickTime and MPEG-4: A Technical Overview	Room A2 **Fri., 3:30pm**

For More Information

- QuickTime Developer Series
  http://developer.apple.com/techpubs/quicktime/qtdevdocs/QT4WebPage/qtdevseries.htm
- QuickTime Developer Documentation
  http://developer.apple.com/techpubs/quicktime/quicktime.html
- Other places
  - **www.blueabuse.com**
  - **www.mactech.com**

http://developer.apple.com/wwdc2002/urls.html

Sample Code

- QuickTime Sample Code
  http://developer.apple.com/samplecode/Sample_Code/QuickTime.htm

Reminder

The QuickTime Engineering Team
Is Holding a “Hands-On Lab” Everyday
From 1:00–4:00pm in Room G . . Stop By!


WWDC2002


WWDC2002


WWDC2002